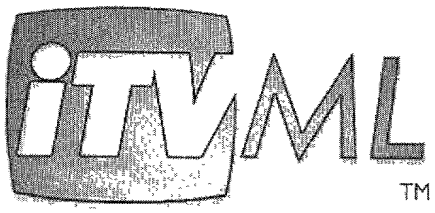


ATTACHMENT A

Interactive Television Markup Language (ITVML)

Draft Specification

Language Version 1.0

March 22, 2001

Copyright Notice

Disclaimer of Warranties and Limitation of Liability

INTELLOCITY USA, INC., ITS AFFILIATES AND SUBSIDIARIES (collectively "INTELLOCITY") MAKE NO EXPRESS OR IMPLIED WARRANTY OF ANY KIND WITH RESPECT TO THE CONTENTS OF THIS DRAFT SPECIFICATION AND SPECIFICALLY DISCLAIM THE IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE. INTELLOCITY SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY, ARISING OUT OF ANY USE OF THIS DRAFT SPECIFICATION. ANY USE OF THIS DRAFT SPECIFICATION INDICATES THE USER'S ACCEPTANCE OF ALL TERMS AND CONDITIONS OF USE SPECIFIED HEREIN.

Table of Contents

1 Introduction to ITVML 1.0

The Interactive Television Markup Language (ITVML) describes device-independent enhanced television content for television programming. By *device*, this specification means set-top boxes (STBs), television signal encoders, and filtering and monitoring equipment. ITVML is a declarative, XML-compliant language.

To create an enhanced television project, a content developer creates an ITVML document, which includes descriptions of interactive content layout, dynamic modifications, and stream-embedded content.

ITVML:

- Enables content developers to create enhanced television content without learning languages and APIs specific to the device
- Reduces time to develop enhanced content
- Provides a standard language for integrating enhanced television applications
- Provides a schema that the iTV industry can use to develop enhanced television applications

1.1 Relationship of ITVML to XML, XSL, and CSS

ITVML complies with the W3C XML 1.0 specification [1]. Appendix A contains the ITVML 1.0 DTD (Document Type Definition).

When ITVML is translated to HTML, CSS style sheets [2] or XSL formatting objects [3] can be used with the resulting HTML. XSLT [4] can be used to transform ITVML to other XML-compliant markup languages.

1.2 Purpose of this Specification

This specification serves as the official language reference for ITVML 1.0 and describes the syntax of the elements and their attributes, the structure of ITVML documents, and provides usage examples. This specification refers to documents that may be helpful when developing applications using ITVML. A list of references appears at the end of this specification.

ITVML is an open, standardized language, which may be implemented freely without any licensing costs. The goal of this draft is to elicit feedback from the iTV industry. Comments are encouraged; please send them to

itvml-editor@itvml.org

After comments are received and incorporated and this draft specification is finalized, it will be submitted to a standards body for consideration.

This specification may be distributed freely, as long as all text and legal notices remain intact.

1.3 Terminology

Terminology used in this specification is defined in this section.

Enhanced Television Content: Interactive content that is embedded as, or referenced by, data inserted into a broadcast video stream.

Viewer: The operator of a set-top box and television.

Platform: The combination of hardware and operating system software that comprises a complete system on which applications can run. Platforms can be a viewer's set-top box configuration or a system that an authoring application runs on.

Encode: The process of inserting data into the VBI (vertical blanking interval) of a broadcast video stream.

1.4 Conventions

A number enclosed in braces refers to standards or other documents listed in References. For example, [5] refers to RFC2119.

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in RFC2119 [5].

Ellipses (...) indicate where attribute values or content have been omitted. Many of the examples in this specification are ITVML fragments. Additional code may be required to develop fully functional content.

URLs included in the code segments in this specification are only for demonstration and may not actually exist.

Table 1 shows the conventions used in this document.

Table 1 - Document Conventions

Convention	Used for
`monospace`	Syntax, examples, user input, and application output
bold	Function names
italic	User supplied parameters
" "	Names or values assigned to objects
[ ]	Optional parameters
|	Either/or choice between two or more items

=	Required value for a parameter
||	Either or both items

2 Document Status

Versions of this specification are available online. The latest version of the ITVML specification is available online at

http://www.itvml.org/docs/itvml

2.1 Editor

Phil Batey
Intellocity USA, Inc.
1400 Market Street
Denver, CO 80202
itvml-editor@itvml.org

2.2 ITVML License Cost

ITVML 1.0 may be implemented freely without any license cost.

2.3 Errata

The list of known errors in this specification is available online at

http://www.itvml.org/docs/itvml10a-errata.html

Please report errors in this specification to

itvml-editor@itvml.org

2.4 Comments

Comments regarding this specification can be submitted by email to

itvml-editor@itvml.org

3 Structure of an ITVML Document

3.1 Overview

An ITVML document is a set of elements that describes the layout of enhanced content and viewer interaction.

In addition, an ITVML document describes the mechanism for delivery and presentation to the viewer.

Each ITVML document includes *content* (images, text, and layout), *actions* (dynamic changes to content), and *triggers* (embedded calls to actions).

3.2 Example of an ITVML Document

Here is an example of an ITVML document that displays the words, "Hello World!".

```
<?xml version="1.0"?>
<!DOCTYPE itvml PUBLIC "-//Intellocity//DTD ITVML 1.0a Draft//EN"
   "http://itvml.org/dtds/ITVML1_0a.dtd">

<itvml>
   <content canvas="640x480">
      <page name="page1">
         <element type="text" name="text1">
            <property name="top" value="120" />
            <property name="left" value="160"/>
            <property name="width" value="240"/>
            <property name="height" value="320"/>
            <property name="value" value="Hello World!"/>
         </element>
      </page>
   </content>
</itvml>
```

3.3 ITVML Document Structure

A typical ITVML 1.0 document is composed of:

- A prolog that identifies the XML language version and encoding and the location of the ITVML1.0 DTD (document type definition)

```
<?xml version="1.0"?>
<!DOCTYPE itvml PUBLIC
   "-//Intellocity/DTD ITVML 1.0a Draft//EN"
   "http://itvml.org/dtds/ITVML1_0a.dtd">
```

Note: This prolog should begin every ITVML document but for readability, some of the examples in this specification omit it.

- The root element in the document, which is the *itvml* tag

```
<itvml xmlns='http://itvml.org/dtds/ITVML1_0a.dtd'> ... </itvml>
```

See Section 5.1 for more information of the root element *itvml*.

The *itvml* element contains the following child elements:

- An optional header element that provides metadata about the project

  `<head> ... </head>`

  The *head* element is discussed in Section 5.2.

- An optional library section, which lists external dependencies

  `<library> ... </library>`

  The *library* element is discussed in Section 6.1.

- An optional description of enhanced content pages

  `<content> ... </content>`

  The *content* element is discussed in Section 7.2.

- An optional description of the timeline of triggers and packages that will be encoded onto the video stream

  `<timeline> ... </timeline>`

  The *timeline* element is discussed in Section 8.1

White space (spaces, new lines, tabs, and XML comments) may appear before or after each of the previously described tags (provided that the XML formatting rules are not violated).

The following example shows a skeleton of an ITVML document.

```
<?xml version="1.0"?>
<!DOCTYPE itvml PUBLIC
   "-//Intellocity//DTD ITVML 1.0a Draft//EN"
   "http://www.itvml.org/dtds/ITVML1_0a.dtd">

<itvml xmlns='http://itvml.org/dtds/ITVML1_0a.dtd'>
   <head> ... </head>
   <library> ... </library>
   <content> ... </content>
   <timeline> ... </timeline>
</itvml>
```

Note: The four elements *head, library, content,* and *timeline* may appear in any order.

3.4 ITVML Namespace

ITVML is designed to function with existing standards. This includes other markup languages that specify platform-dependent formatting (HTML for text, JSGF for voice).

XML Namespaces eliminate the problem of recognition and collisions between elements and attributes of two or more markup vocabularies in the same file.

All ITVML elements and attributes are in the *itvml* namespace, identified by the URI, http://itvml.org/dtds/ITVML1_0a.dtd

3.4.1 ITVML Namespace Example

The following example combines ITVML and HTML vocabularies.

```
<itvml:itvml xmlns:itvml='http://itvml.org/dtds/ITVML1_0a.dtd'>
   <itvml:content>
      <itvml:page>
         <itvml:element type="text">
            <itvml:property name="value">
               <html:em xmlns:html='http://www.w3.org/TR/REC-html40'
                  >Emphasis</html:em>
            </itvml:property>
         </itvml:element>
      </itvml:page>
   </itvml:content>
</itvml:itvml>
```

This code can be simplified by making itvml the default namespace, as in the following example.

```
<itvml xmlns='http://itvml.org/dtds/ITVML1_0a.dtd'>
   <content>
      <page>
         <element type="text">
            <property name="content">
               <html:em xmlns:html='http://www.w3.org/TR/REC-html40'
                  >Emphasis</html:em>
            </property>
         </element>
      </page>
   </content>
</itvml>
```

3.5 ITVML Mime Type

The following mime (Multipurpose Internet Mail Extensions) type should be used for ITVML documents. Mime is described in RFC2045, RFC2046, RFC2047, RFC2048, and RFC2049 [8].

```
text/itvml
```

3.6 ITVML File Extension

When an ITVML document is stored as a file the preferred file extension is ".itv".

4 Table of ITVML Elements

Table 2 provides an overview of elements in ITVML and an index to where they are discussed in this specification. The ITVML 1.0 DTD is in Appendix A – ITVML 1.0 Document Type Definition.

Table 2 - ITVML Elements

Element	Purpose	Page
<action>	A collection of property modifications that are performed together	22
<action-call>	A call to an action from an action	23
<component>	A file to be broadcast	29
<content>	Describe the presentation of the enhanced content	12
<element>	An item that appears on a page	13
<head>	Information about the document	10
<itvml>	Top-level element in each ITVML document	9
<library>	List of external dependencies	11
<meta>	Information about the ITVML document	10
<modify-property>	Specify a property modification	23
<page>	A collection of elements that are displayed to the viewer	13
<parameter>	Specifies a parameter of an action	22
<payload>	Broadcast package that contains components	28
<property>	Specifies an element property	21
<resource>	An collection of assets referenced by elements in content	11
<script>	Embedded scripting language	25
<sync-data>	Synchronized data	29
<timed-action-call>	An call to an action that is timed relative to page load	23
<timeline>	Timing for embedded content and triggers	26
<trigger>	A call to an action or script that is timed relative to the broadcast video	27

The elements of the ITVML 1.0 DTD are shown in Figure 1. Reference relationships between ITVML elements are shown with dashed lines.

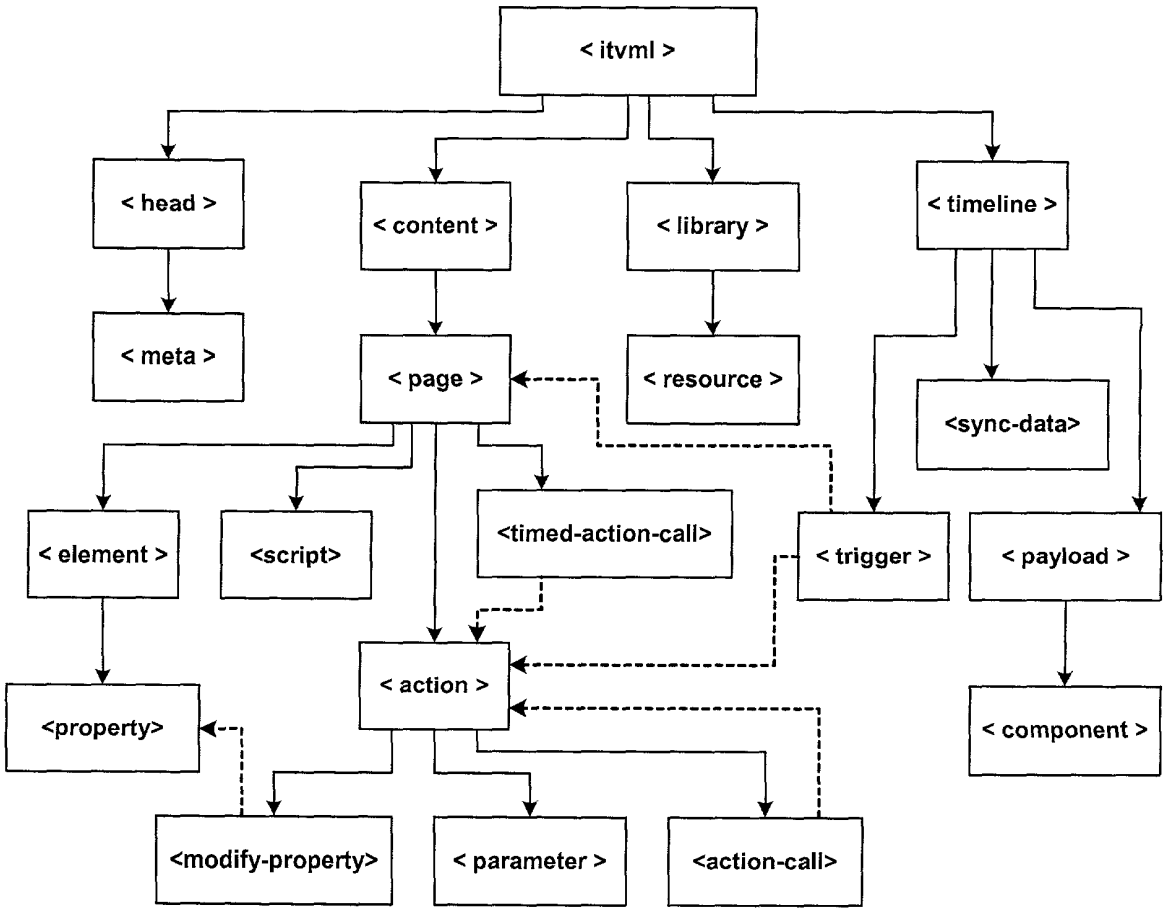


Figure 1 - ITVML 1.0 Document Type Definition

5 The *itvml* and *head* Elements

Whenever a new element is introduced in this specification, the appropriate DTD fragment is shown.

5.1 The *itvml* Element

DTD

```
<!ELEMENT itvml (head?, library?, content?, timeline?)>
```

Description

The *itvml* element is the root element in an ITVML document. All other elements are contained in the *itvml* element.

The *itvml* element appears as

```
<itvml>...</itvml>
```

Usually, one *itvml* element equates to one file, in much the same way that there is one HTML element per file when developing HTML-based documents.

When multiple markup vocabularies are used in the same ITVML file, the *itvml* namespace must be specified as

```
<itvml xmlns='http://itvml.org/dtds/ITVML1_0a.dtd'>...</itvml>
```

5.2 The *head* Element

DTD

```
<!ELEMENT head (meta)*>
```

Description

The *head* element contains information about the current ITVML document. Elements in the head element are not considered part of the enhanced content and have no effect on the presentation or operation of the content.

ITMVL authoring tools should use the *head* element to store information about the document (for example, author, date, and version) and other proprietary information.

5.2.1 The *meta* Element

DTD

```
<!ELEMENT meta EMPTY>
<!ATTLIST meta
   name      NMTOKEN  #REQUIRED
   content   CDATA    #REQUIRED>
```

Description

The *meta* element uses the same semantics as the meta element in HTML. It describes a single piece of information about the current ITVML document. This element may include author information, date of creation, or other information.

The *name* attribute specifies the meta-information; the *content* attribute is content.

Example

```
<head>
   <meta name="Author" content="ITVML Editor"/>
   <meta name="Date-created" content="16 Feb 2001"/>
   <meta name="Notes" content=
      "This is an example of how to use the meta tag in ITVML.
      The content of the meta tag can include white space."/>
   <meta name="Project-name" content="ITVML Example"/>
   <meta name="Project-expires" content="25 Dec 2001 21:00:00 MST"/>
</head>
```

6 Library Description

This section describes the elements of the *library* element, their attributes, and syntax. Examples are provided to help show common usage of each element.

6.1 The *library* Element

DTD

```
<!ELEMENT library (resource)*>
```

Description

The *library* element contains descriptions of asset containers. These containers are external resources required for presenting and encoding the enhanced content.

6.1.1 The *resource* Element

DTD

```
<!ELEMENT resource EMPTY>
<!ATTLIST resource
   name      NMTOKEN  #IMPLIED
   src       CDATA    #IMPLIED>
```

Description

The *resource* element describes the URL of a container of assets that can be referenced by *content* elements.

The *name* attribute specifies a unique identifier. The *source* attribute defines the location of the resource.

Assets in a container can be referenced by the library name by using the following syntax.

```
library://library-name/asset
```

Example

```
<library>
   <resource name="assetDir" src="../assets"/>
   <resource name="assetURL" src="http://itvml.org/assets"/>
   <resource name="assetFile" src="assets.zip"/>
</library>
```

7 Content Description

This section describes the elements of the *content* element, their attributes, and syntax. Examples show common usage for each element.

7.1 Overview

The *content* element contains *page* elements, which in turn contain *element, action*, *script*, and *timed-action-call* elements.

```
<content>
   <page>
      <element>
         <property/>
      </element>
      <action>
         <action-call/>
         <modify-property/>
      </action>
      <timed-action-call/>
      <script/>
   </page>
</content>
```

An *element* element is a TV-viewable item, which is displayed to the viewer. A *property* describes presentation attributes that are fixed or can be modified. An *action* groups a list of *property* changes that are to be executed together. A *timed-action-call* specifies a call to an *action* at a time relative to the page being loaded.

7.2 The *content* Element

DTD

```
<!ELEMENT content (page)*>
<!ATTLIST content
   canvas      CDATA    '640x480'
   pixelAspect CDATA    '1:1'
   baseURL     CDATA    #REQUIRED>
```

Description

The *content* element describes general attributes of all contained pages.

The *canvas* attribute describes the resolution in pixels of a single *page* of television content measured to include full television overscan. The *canvas* attribute is specified as *width* x *height*.

The *pixelAspect* attribute enables non-square pixels (for example, some authoring tools use an D1 NTSC canvas size of 720x486 with a pixel aspect ratio of .9:1). The *pixelAspect* is specified as *width* x *height*.

The *baseURL* attribute specifies the location that client-browsers will use to access the pages. A client-browser references contained pages by combining the *baseURL* and the page name. For example, if a *baseURL* is specified as "`http://www.itvml.org/examples`" and a page is named "page1" the full URL of the page is "`http://www.itvml.org/examples/page1.html`".

7.2.1 The *page* Element

DTD

```
<!ELEMENT page (element|action|timed-action-call|script)*>
<!ATTLIST page
   name      NMTOKEN  #IMPLIED>
```

Description

A *page* contains a list of *elements* that are displayed to the viewer, *actions* that describe modifications to *elements*, and *timed-actions* that specify when actions should occur.

7.2.2 The *element* Element

DTD

```
<!ENTITY % ElementTypes
   "(image|text|tv|field|html)">

<!ELEMENT element (property)*>
<!ATTLIST element
   name      NMTOKEN          #IMPLIED
   type      %ElementTypes;   #REQUIRED
   target    CDATA            'any'>
```

Description

An *element* is an object that is displayed to the viewer. The *type* attribute defines how and whether the viewer can interact with the *element*. The *name* must uniquely identify the *element* on a *page*.

The *target* is a space-separated list of interactive television platforms that the application queries to determine whether the element displays. A *target* of "any" indicates that the target will be displayed on all platforms.

An *element* contains a list of fixed or modifiable *properties*. The specific list of *properties* of an *element* depends on the *type*.

Example

```
<element name="tv-object" type="tv">
   <property name="width" value="100">
   <property name="height" value="75">
   <property name="top" value="0">
   <property name="left" value="565">
</element>
```

7.2.2.1 The *image* Element Type

Table 3 - *image* Element Properties

Property	Description	Required
width	The width of the area to display the image; *Positive integer*	Yes
height	The height of the area to display the image; *Positive integer*	Yes
top	Offset from the top of the page; *Positive integer*	Yes
left	Offset from the left side of the page; *Positive integer*	Yes
src	The URL of the image to display; *URL*	Yes
pageref	A link to another page in the content section; *String*	No
action	Call an action when the image is selected; *String*	No
href	The URL that the image links to; *URL*	No
script	A call to a function within a script element; *String*	No
zOrder	The stacking order relative to other elements; *Positive integer; default: 0*	No
highlight	A default highlight effect is used; *Boolean; default: true*	No
highlightSrc	An image source to use when the cursor is focused on this element; *URL*	No
transparency	A percentage of transparency; *0-100; default: 0*	No

Description

The *image* element type displays a graphic specified by the *src* attribute in an area (*width, height, top, left*).

Note: The origin (0, 0) of the canvas is located at the top left corner. The numerical values displayed in the Top and Left properties are positive numbers, indicating the distance from the origin to the top left corner of the selected graphic. The position of the bottom right corner of the NTSC canvas is (640, 480).

If a *pageref, action, href,* or *script* is specified, the image is made "active". An "active" image can be selected by the user to execute an *action* or change the currently viewed page. Only one of either *pageref* or *href* may be specified. If both *action* and *script* are specified, they will be executed sequentially; the *action* first, then the *script*. *action* and *script* will be executed before the current page is changed by a specified *pageref* or *href*.

A reference to an *action* must be in the form of a function call with any *parameters* described in the definition of the *action* (for example, `action(1,2,"string-val")` or `action()` if no *parameters* are defined).

In addition to actions specified in the content, the submitPage() action can be specified. The **submitPage()** action takes as a parameter an href of a CGI program. All *field* elements will be submitted to the CGI program. For example, `submitPage("http://www.itvml.org/cgi-bin/regform.cgi")` calls the URL, `http://www.itvml.org/cgi-bin/regform.cgi?field=value&field2=val2`.

If the cursor is moved to an "active" element, that element becomes highlighted as the element of focus. Setting *highlight* to true indicates the default highlight effect will be used. If the *highlightSrc* property is specified, the element will display the indicated image. When the element loses focus, the displayed image reverts to the *src* image.

zOrder indicates the relative stacking order of elements. Elements with a higher numbered *zOrder* will obscure elements with a lower *zOrder*. Overlapping elements with matching *zOrder* will cause undefined behavior. *zOrder* is specified as a positive integer.

transparency indicates the percent of obscured elements that should show through the image.

Example

```
<element type="image" name="image1">
   <property name="top" value="0"/>
   <property name="left" value="0"/>
   <property name="width" value="100"/>
   <property name="height" value="100"/>
   <property name="src" value="image.jpg"/>
   <property name="highlightSrc" value="image_highlight.jpg"/>
   <property name="pageref" value="page2"/>
```

</element>

7.2.2.2 The *text* Element Type

Table 4 - *text* Element Properties

Property	Description	Required
width	The width of the area in which to display the text; *Positive integer*	Yes
height	The height of the area in which to display the text; *Positive integer*	Yes
top	Offset from the top of the page; *Positive integer*	Yes
left	Offset from the left side of the page; *Positive integer*	Yes
font	The font to use when displaying text; *String (See* Appendix B – Font Style*)*	No
align	The alignment of the text within the area; *String (See* Figure 2*)*	No
value	The string to display in the element; *String*	No
bgcolor	The color to use as the background of the area; *Color as hex value*	No
gradcolor	Gradient color to transition from the bgcolor; *Color as hex value*	No
gradangle	Gradient angle; *Integer; default: 0*	No
background	Background image; *URL*	No
visible	Is this element displayed; *Boolean; default: true*	No
transparency	A percentage of transparency; *0-100; default: 0*	No
zOrder	The stacking order relative to other elements; *Positive integer; default: 0*	No

Description

The *text* element type displays a text string in an area (*width, height, top, left)*. The *font* property specifies how the *text* element is displayed.

The *font* property value follows the CSS2 [2] syntax for the *font* style attribute. See *Appendix B – Font* Style for details about font style.

The *align* property can be set to one of the following values: *top, top-left, top-right, right, center, left, bottom, bottom-left, bottom-right*. The default *align* value is *left*. The alignment values are described in *Figure 2*.

top-left	top	top-right
left	center	right
bottom-left	bottom	bottom-right

Figure 2 - Alignment Values

The *value* of the *text* element specifies the actual text string to display in the area.

The *bgcolor, gradcolor, gradangle, background*, and *transparency* properties indicate how the background of the area should be displayed.

bgcolor and *gradcolor* are specified as hex color values. If *bgcolor* is specified without a *gradcolor*, the background will be displayed as a solid color.

If *gradcolor* is specified, a gradient between the two colors will be generated for the background.

gradangle indicates the angle of the transition between *bgcolor* and *gradcolor;* an angle of zero indicates a left to right gradient; 90 indicates a bottom to top gradient; 180 indicates right to left; 270 indicates top to bottom.

If the *background* property is specified, *bgcolor, gradcolor, and gradangle* are ignored. The indicated image is displayed as the background for this element.

transparency indicates the percentage of obscured elements that should show through the background of this element.

The *visible* property indicates that the element should be displayed.

Example

```
<element type="text" name="text1">
   <property name="top" value="100"/>
   <property name="left" value="160"/>
   <property name="width" value="320"/>
   <property name="height" value="200"/>
   <property name="align" value="center"/>
   <property name="font" value="20pt bold Arial"/>
   <property name="value" value="Hello World!"/>
   <property name="background" value="textbg.jpg"/>
</element>
```

7.2.2.3 The *tv* Element Type

Table 5 - *tv* Element Properties

Property	Description	Required
width	The width of the area to display the tv object; *Positive integer*	Yes
height	The height of the area to display the tv object; *Positive integer*	Yes
top	Offset from the top of the page; *Positive integer*	Yes
left	Offset from the left side of the page; *Positive integer*	Yes
channel	Channel that the tv object is tuned to; *String*	No
pageref	A link to another page in the content section; *String*	No
action	Call an action when the tv is selected; *String*	No
href	The URL that the tv should link to; *URL*	No
script	A call to a function within a script element; *String*	No
zOrder	The stacking order relative to other elements; *Positive Integer; default: 0*	No
highlight	A highlight effect is used; *Boolean; default: true*	No
visible	The element is displayed; *Boolean; default: true*	No

Description

The *tv* element type describes an area where the broadcast video is displayed. Only one *tv* element can display on a page.

The *channel* property indicates the video source to which the tv element is tuned. The *channel* is specified as a string that refers to a defined channel by name or number.

If the television *width* and *height* properties match the canvas size of the *content* element, the *tv* element will be displayed in the background, ignoring *zOrder* value. A *tv* element displayed in the background cannot be "active" and will ignore *pageref, href, action*, or *script* properties.

Otherwise, properties behave the same as the image properties described in section 7.2.2.1, *The image Element Type.*

Example

```
<element type="tv">
   <property name="width" value="640"/>
   <property name="height" value="480"/>
   <property name="top" value="0"/>
   <property name="left" value="0"/>
</element>
```

7.2.2.4 The *field* Element Type

Table 6 - *field* Element Properties

Property	Description	Required
width	The width of the area to display the field; *Positive integer*	Yes
height	The height of the area to display the field; *Positive integer*	Yes
top	Offset from the top of the page; *Positive integer*	Yes
left	Offset from the left side of the page; *Positive integer*	Yes
align	Determines where in the element area the field should be displayed; *String (see text element description)*	No
value	The value of the field; *String*	No
maxLength	The maximum number of characters allowed in the field; *Positive integer*	No
font	The font to use when displaying the field; *String (see appendix B)*	No
required	A flag indicating that the field is required; *Boolean; default: false*	No
regexp	A regular expression that the field contents are required to match; *String (See appendix C)*	No
bgcolor	The color to use as the background of the area; *Color as hex value*	No
gradcolor	Gradient color to transition from the bgcolor; *Color as hex value*	No
gradangle	Gradient angle; *Integer; default: 0*	No
background	Background image; *URL*	No
transparency	A percentage of transparency; *0-100; default: 0*	No
zOrder	The stacking order relative to other elements; *Positive integer; default: 0*	No
visible	The element is displayed; *Boolean; default: true*	No

Description

The *field* element type displays an input box to the viewer in the specified area. The values of all *field* elements are submitted to a URL by specifying "**submitPage(***href***)**" as the *action* property of an *image* element type.

The *required, regexp, and maxLength* properties indicate that before field values can be transmitted to a CGI program, they first must be validated. *required* indicates that the field cannot be empty, *maxLength* indicates that the field cannot exceed a number of characters, and *regexp* describes a regular expression that the field value must match. See *Appendix C – Regular Expressions* for details.

Otherwise, properties behave in the same manner as the text properties described in section 7.2.2.2, *The text Element Type.*

Example

```
<element type="field" name="zip">
   <property name="width" value="100"/>
   <property name="height" value="25"/>
   <property name="top" value="200"/>
   <property name="left" value="300"/>
   <property name="regexp" value="/^\d{5}(-\d{4})?$/"/>
   <property name="required" value="true"/>
</element>
```

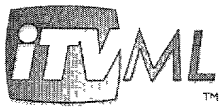
7.2.2.5 The *html* Element Type

Table 7 - *html* Element Properties

Property	Description	Required
width	The width of the area to display the field; *Positive integer*	Yes
height	The height of the area to display the field; *Positive integer*	Yes
top	Offset from the top of the page; *Positive integer*	Yes
left	Offset from the left side of the page; *Positive integer*	Yes
content	The html content to include; *html content*	Yes
visible	The element is displayed; *Boolean; default: true*	No
src	The location of the html to include in the element.	No

Description

The *html* element enables HTML content to be included in an ITVML document. The html displayed in the element can be included in the ITVML document with the *content* attribute or the *html* element can reference external content with the *src* attribute.



Example

```
<element type="html">
   <property name="top" value="160"/>
   <property name="left" value="120"/>
   <property name="width" value="320"/>
   <property name="height" value="240"/>
   <property name="content">
      <table>
      <tr><th>Name</th><th>Title</th></tr>
      <tr><td>Guy Smily</td><td>Software Hack</td></tr>
      <tr><td>Joe Schmoe</td><td>Market Monkey</td></tr>
      </table>
   </property>
</element>

<element type="html">
   <property name="top" value="160"/>
   <property name="left" value="120"/>
   <property name="width" value="320"/>
   <property name="height" value="240"/>
   <property name="src" value="contacts.html"/>
</element>
```

7.2.3 The *property* Element

DTD

```
<!ELEMENT property (#PCDATA)>
<!ATTLIST property
   name      NMTOKEN  #REQUIRED
   value     CDATA    #IMPLIED>
```

Description

A *property* of an *element* is a single attribute that determines how that *element* is displayed or interacts with the viewer. A *property* of an *element* can be modified by an *action.*

Example

```
<element name="image1" type="image">
   <property name="src" value="image1.gif"/>
   <property name="width" value="200"/>
   <property name="height" value="150"/>
   <property name="left" value="440"/>
   <property name="top" value="0"/>
</element>

<action name="swapImage">
   <modify-property element="image1" name="src" value="image2.gif"/>
</action>
```

7.2.4 The *action* Element

DTD

```
<!ELEMENT action (parameter|modify-property|action-call)*>
<!ATTLIST action
   name      NMTOKEN  #REQUIRED>
```

Description

An *action* lists a set of modifications to *element properties* by specifying *modify-property* elements.

An *action* can reference another *action* by including an *action-call* element. The *value* attribute of an *action-call* element cannot be a circular reference by the containing *action.*

Example

```
<action name="moveToTopLeft">
   <modify-property element="image1" name="top" value="0"/>
   <modify-property element="image1" name="left" value="0"/>
</action>

<action name="moveImage">
   <parameter name="top"/>
   <parameter name="left"/>
   <modify-property element="image1" name="top" value="top"/>
   <modify-property element="image1" name="left" value="left"/>
</action>

<element type="image" name="image1">
   <property name="src" value="button.jpg"/>
   <action-call value="moveImage(100,200)"/>
</element>

<action name="moveAndSwap">
   <action-call value="moveImage(50,100)"/>
   <modify-property element="image1"
      name="src" value="button1.jpg"/>
</action>
```

7.2.5 The *parameter* Element

DTD

```
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
   name  NMTOKEN  #REQUIRED>
```

Description

The *parameter* element specifies a value passed to the *action*. The *parameter* can be used as a *value* in a property modification that is contained in the *action*. When a *timed-action-call* or *trigger* refers to the *action*, the caller must specify a value for each *parameter*.

Example

```
<element name="text" type="text">
   <property name="value" value="Old Text"/>
</element>
<action name="updateText">
   <parameter name="newText"/>
   <modify-property element="text" name="value" value="newText"/>
</action>
<timed-action-call startSeconds="10"
   action="updateText('New Text')"/>
```

7.2.6 The *modify-property* Element

DTD

```
<!ELEMENT modify-property (#PCDATA)>
<!ATTLIST modify-property
   element  NMTOKEN  #REQUIRED
   name     NMTOKEN  #REQUIRED
   value    CDATA    #REQUIRED>
```

Description

A *modify-property* indicates a change to an element by an action. The *element* indicates the element on the page that is to be modified. The *name* indicates the *property* of the element to affect. The *value* indicates the new *property* value.

Example

```
<element name="img" type="image">
   <property name="src" value="old_image.jpg"/>
</element>
<action name="update">
   <modify-property element="img" name="src" value="new_image.jpg"/>
</action>
```

7.2.7 The *action-call* Element

DTD

```
<!ELEMENT action-call EMPTY>
<!ATTLIST action-call
   value    CDATA    #REQUIRED>
```

Description

An *action-call* element indicates a call to another action by an action. The *value* attribute indicates the action to call.

A reference to an *action* must be in the form of a function call with all *parameters* described in the definition of the *action* (for example, `action(1,2,'string-val')` or `action()` if no *parameters* are defined).

Example

```
<element type="image" name="image1">
   <property name="src" value="button.jpg"/>
   <action-call value="moveImage(100,200)"/>
</element>

<action name="moveTL">
   <modify-property element="image1" name="top" value="0"/>
   <modify-property element="image1" name="left" value="0"/>
</action>

<action name="moveAndSwap">
   <action-call value="moveTL()"/>
   <modify-property element="image1"
      name="src" value="button1.jpg"/>
</action>
```

7.2.8 The *timed-action-call* Element

DTD

```
<!ELEMENT timed-action-call EMPTY>
<!ATTLIST timed-action-call
   name                NMTOKEN   #IMPLIED
   value               CDATA     #REQUIRED
   startTime           CDATA     #REQUIRED
   loopNTimes          CDATA     "1"
   loopInterval        CDATA     #IMPLIED>
```

Description

A *timed-action-call* is a call to an *action* or function in an included *script* that is timed relative to the page invocation and not tied to the timeline of the broadcast video, like a *trigger*.

The attributes *startTime, loopNTimes*, and *loopInterval* determine the starting point and frequency of the *action* specified by the *value* attribute. If *loopNTimes* is set to "0", the *action* specified in the *value* is repeated until the viewer navigates to another page. *startTime* and *loopInterval* are specified in seconds.

A reference to an *action* must be in the form of a function call with any *parameters* described in the definition of the *action* (for example, `action(1,2,'string-val')` or `action()` if no *parameters* are defined).

Example

```
<timed-action-call name="adbanner"
   description="swap the banner ad every 15 seconds"
   value="swapAd()"
   startTime="15"
   loopNTimes="0"
   loopInterval="15">
```

7.2.9 The *script* Element

DTD

```
<!ELEMENT script CDATA>
<!ATTLIST script
   language CDATA    #IMPLIED
   target   CDATA    #IMPLIED
   src      CDATA    #IMPLIED>
```

Description

A *script* element enables the use of a scripting language, such as javaScript, to perform calculations.

The *target* attribute is a space-separated list of strings that filter the use of the script only to listed platforms. If *target* is not specified or specified as "any", the script will be used on all platforms.

The *language* attribute specifies the language of the script.

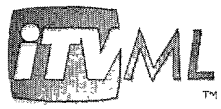
The script can be included in the ITVML document by embedding the source code in the *script* element or the script may refer to an external file with the *src* attribute.

Example

```
<script language="javascript" target="any">
function updateTime()
{
   now = new Date();
   // call an action to update text
   setTimeValue(now);
}
</script>

<script language="javascript" target="any" src="updateTime.js"/>
```

8 Timeline Description

This section describes the elements of the *timeline* element, their attributes, and syntax. Examples show common usage for each element.



8.1 Overview

A *timeline* element describes the timing of *triggers* and *payloads* to be encoded into a video stream. A *payload* consists of one or more *components.*

```
<timeline>
   <trigger/>
   <payload>
      <component/>
   </payload>
   <sync-data/>
</timeline>
```

8.2 The *timeline* Element

DTD

```
<!ELEMENT timeline (trigger|payload)*>
<!ATTLIST timeline
   length            CDATA          #IMPLIED
   framesPerSecond   CDATA          #IMPLIED
   usesDropFrame     (true|false)   "false"
   videoStartTime    CDATA          #IMPLIED
   loopNTimes        CDATA          "1">
```

Description

The *timeline* is the list of *triggers* and *payloads* and describes when they will be inserted into the broadcast video.

The attribute *length* describes the length of the timeline to be used with a broadcast.

The attributes *framesPerSecond* and *usesDropFrame* affect the translation between times and frame counts. *usesDropFrame* is ignored unless *framesPerSecond* is set to 30, where two frame counts are dropped every minute, except at every 10 minutes where none are dropped. This calculation matches the NTSC frame rate of 29.97 frames per second. As PAL and SECAM frame rates are 25fps, *useDropFrame* has no effect.

Note: This calculation affects frame counts only, no video frames are removed.

The attribute *videoStartAtTime* specifies the offset into the video stream to begin the timeline.

loopNTimes specifies the number of times to repeat the timeline; a setting of '0' indicates that the timeline should be repeated until the end of the timeline.

The attributes *length* and *videoStartTime* are specified as SMPTE time-codes; hours, followed by 0-59 minutes, then 0-59 seconds, and a number of frames.

hours:minutes:seconds.frames

Example

```
<timeline
   length="01:00:00"
   videoStartAtTime="00:00:10.5"
   framesPerSecond="30">
   <!-- triggers go here... -->
</timeline>
```

8.2.1 The *trigger* Element

DTD

```
<!ELEMENT trigger EMPTY>
<!ATTLIST trigger
   name              NMTOKEN         #IMPLIED
   title             CDATA           #IMPLIED
   startTime         CDATA           #IMPLIED
   href              CDATA           #IMPLIED
   script            CDATA           #IMPLIED
   pageref           NMTOKEN         #IMPLIED
   action            CDATA           #IMPLIED
   repeatNTimes      CDATA           "1"
   loopNTimes        CDATA           "1"
   loopInterval      CDATA           #IMPLIED
   enabled           (true|false)    "true"
   expiration        CDATA           #IMPLIED>
```

Description

A *trigger* is a call to an *action, script*, *href*, or a page described in the content that is timed relative to the timeline of the broadcast video. Trigger timing is *not* relative to the page loading, like a *timed-action-call*.

The *title* is the text displayed to the viewer when the *trigger* is broadcast.

Either *pageref* or *href* may be specified. If both *action* and *script* are specified, they will be executed sequentially; the *action* first, then the *script*.

The attributes *startTime, repeatNTimes*, *loopNTimes*, and *loopInterval* determine the starting point and frequency of the *trigger*. The *trigger* is repeated for the entire timeline if *loopNTimes* is set to "0". *repeatNTimes* specifies the number of times in succession that the trigger should be encoded.

startTime and *loopInterval* are specified in SMPTE time-codes; hours, followed by 0-59 minutes, then 0-59 seconds, and a number of frames.

hours:minutes:seconds.frames

The *expiration* of a *trigger* indicates that the client browser should ignore the trigger after a specified time. *expiration* is specified using the ISO-8601 date/time format (for example: 2001-03-17T13:10:30Z is 17 March 2001, 10

minutes and 30 seconds past one pm), with the exception that UTC is assumed.

Example

```
<trigger name="trivia1"
   title="Answer Trivia Question"
   href="trivia.html"
   script="showTrivia1()"
   time="00:03:50.15"/>
```

8.2.2 The *payload* Element

DTD

```
<!ELEMENT payload (component)*>
<!ATTLIST payload
   name                 NMTOKEN        #IMPLIED
   startTime            NMTOKEN        #REQUIRED
   expires              CDATA          #REQUIRED
   integrityCheck       (true|false)   'false'
   baseURL              CDATA          #IMPLIED
   retransmitExpiration CDATA          #IMPLIED>
```

Description

A *payload* is the actual data that is embedded in the video stream. A *payload* consists of one or more *components.*

The *integrityCheck* flag indicates that data integrity measures should be used. For example, in the case of ATVEF, the CRC checksum should be included as described in the ATVEF Specification [6].

The *baseURL* attribute indicates that all *components* should be cached with reference to this URL.

The *startTime* attribute indicates the point in the timeline when the payload should be inserted into the broadcast. *startTime* is specified in SMPTE time codes; hours, followed by 0-59 minutes, then 0-59 seconds, and a number of frames

hours:minutes:seconds.frames

The *expires* attribute indicates the date and time after which the *payload* cannot be used. *expiration* is specified using the ISO-8601 date/time format (for example: 2001-03-17T13:10:30Z is 17 March 2001, 10 minutes and 30 seconds past one pm in UTC), with the exception that UTC is assumed.

The *retransmitExpiration* indicates how long the client should wait for another transmission to fill in data that might have been corrupted.

Example

```
<payload startTime="00:02:04.3" expires="2001-02-04"
   baseURL="lid:/cache/itvml/">
   <component source="image.gif" location="image2.gif"/>
</payload>
```

8.2.3 The *component* Element

DTD

```
<!ELEMENT component EMPTY>
<!ATTLIST component
   name      NMTOKEN  #IMPLIED
   src       CDATA    #REQUIRED
   location  CDATA    #IMPLIED
   expires   CDATA    #IMPLIED>
```

Description

A *component* describes a file to be embedded into the video stream. The *source* indicates where the source file exists, and the *location* indicates the name that the set-top box client should use to store and reference the file.

If *location* is not specified, the filename portion of the *source* attribute will be used.

The *expires* attribute indicates the date and time after which the *component* should not be used. *expiration* is specified using the ISO-8601 date/time format (for example: 2001-03-17T13:10:30Z is 17 March 2001, 10 minutes and 30 seconds past one pm), with the exception that UTC is assumed.

Example

```
<payload startTime="00:02:04.3" expires="2001-02-04">
   <component source="image.gif" location="image2.gif"/>
</payload>
```

8.2.4 The *sync-data* Element

DTD

```
<!ELEMENT sync-data EMPTY>
<!ATTLIST sync-data
   startTime   NMTOKEN                          #REQUIRED
   field       (1|2)                            #IMPLIED
   type        (C1|C2|C3|C4|T1|T2|T3|T4|XDS)    #REQUIRED
   content     CDATA                            #REQUIRED>
```

Description

A *sync-data* element describes synchronous data such as closed captioning that is embedded in the video.

The *startTime* attribute indicates at what point in the timeline the payload should be inserted into the broadcast. *startTime* is specified in SMPTE time codes; hours, followed by 0-59 minutes, then 0-59 seconds, and a number of frames

hours:minutes:seconds.frames

The *field* indicates the field of a frame of interlaced video on which the data is embedded. *field* applies only to interlaced video sources.

The *type* of the sync-data element indicates how the content will be used. C1, C2, C3, and C4 indicate closed captioning; T1, T2, T3, and T4 indicate text for uses other then closed captioning; XDS indicates v-chip data for parental control.

The *content* attribute contains the text that will be embedded in the broadcast.

Example

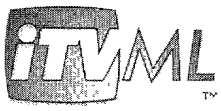
```
<sync-data startTime="00:02:04.3" field="1" type="C1"
   content="Launch in 5..."/>
<sync-data startTime="00:02:05.3" field="1" type="C1"
   content="4..."/>
<sync-data startTime="00:02:06.3" field="1" type="C1"
   content="3..."/>
<sync-data startTime="00:02:07.3" field="1" type="C1"
   content="2..."/>
<sync-data startTime="00:02:08.3" field="1" type="C1"
   content="1..."/>
<sync-data startTime="00:02:09.3" field="1" type="C1"
   content="Blast Off!"/>
```

Appendix A – ITVML 1.0 Document Type Definition

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--
    Interactive Television Markup Language (ITVML)
    ==============================================
    Developed by:

        Intellocity USA, Inc.

    Authors:

        Phil Batey (phil@intellocity.com)

    Usage:

        <?xml version="1.0"?>
        <!DOCTYPE itvml PUBLIC
            "-//Intellocity//DTD ITVML 1.0a Draft//EN"
            "http://itvml.org/dtds/ITVML1_0a.dtd">
        <itvml>
            <head> ... </head>
            <library> ... </library>
            <content> ... </content>
            <timeline> ... </timeline>
        </uiml>

    Description:

        This DTD corresponds to the UIML 2.0a specification,
        which may be found at the following URL:
        http://www.itvml.org/docs/itvml10

    Change History:

        22 Feb 2001 - P Batey (phil@intellocity.com)
                      - first draft
-->

<!ELEMENT itvml (head?, library?, content?, timeline?)>

<!ELEMENT head (meta)*>

<!ELEMENT meta EMPTY>
<!ATTLIST meta
    name          NMTOKEN     #REQUIRED
    content       CDATA       #REQUIRED>

<!ELEMENT library (resource)*>
```

```
<!ELEMENT resource EMPTY>
<!ATTLIST resource
    name            NMTOKEN     #IMPLIED
    src             CDATA       #IMPLIED>

<!ELEMENT content (page)*>
<!ATTLIST content
    canvas          CDATA       #REQUIRED
    baseURL         CDATA       #REQUIRED>

<!ELEMENT page (element|action|timed-action-call|script)*>
<!ATTLIST page
    name            NMTOKEN     #IMPLIED
    location        CDATA       #REQUIRED>

<!ENTITY % ElementTypes
    "(image|text|tv|field|html)">

<!ELEMENT element (property)*>
<!ATTLIST element
    name            NMTOKEN             #IMPLIED
    type            %ElementTypes;      #REQUIRED
    target          CDATA               #IMPLIED>

<!ELEMENT property (#PCDATA)>
<!ATTLIST property
    name            NMTOKEN     #REQUIRED
    value           CDATA       #IMPLIED
    element         NMTOKEN     #IMPLIED>

<!ELEMENT action (parameter|modify-property|action-call)*>
<!ATTLIST action
    name            NMTOKEN     #REQUIRED
    params          CDATA       #IMPLIED>

<!ELEMENT modify-property (#PCDATA)>
<!ATTLIST modify-property
    element         NMTOKEN     #REQUIRED
    name            NMTOKEN     #REQUIRED
    value           CDATA       #IMPLIED>

<!ELEMENT action-call EMPTY>
<!ATTLIST action-call
    value           CDATA       #REQUIRED>

<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
    name            NMTOKEN     #REQUIRED>

<!ELEMENT timed-action-call (parameter)*>
<!ATTLIST timed-action-call
    name            NMTOKEN     #IMPLIED
    description     CDATA       #IMPLIED
    value           CDATA       #REQUIRED
```

```
    startSeconds  CDATA       #REQUIRED
    loopNTimes    CDATA       "1"
    loopInterval  CDATA       #IMPLIED>

<!ELEMENT script CDATA>
<!ATTLIST script
    target        CDATA       #IMPLIED
    language      CDATA       #IMPLIED
    src           CDATA       #IMPLIED>

<!ELEMENT timeline (trigger|payload)*>
<!ATTLIST timeline
    lengthInFrames    CDATA         #IMPLIED
    videoFilename     CDATA         #IMPLIED
    videoTime         CDATA         #IMPLIED
    framesPerSecond   CDATA         #IMPLIED
    usesDropFrame     (true|false)  "false"
    loopNTimes        CDATA         "1">

<!ELEMENT trigger EMPTY>
<!ATTLIST trigger
    name              NMTOKEN       #IMPLIED
    description       CDATA         #IMPLIED
    startTime         CDATA         #IMPLIED
    expiration        CDATA         #IMPLIED
    href              CDATA         #IMPLIED
    script            CDATA         #IMPLIED
    pageref           CDATA         #IMPLIED
    action            CDATA         #IMPLIED
    repeatNTimes      CDATA         "1"
    loopNTimes        CDATA         "1"
    loopInterval      CDATA         "0"
    enabled           (true|false)  "true">

<!ELEMENT payload (component)*>
<!ATTLIST payload
    name                  NMTOKEN       #IMPLIED
    startTime             NMTOKEN       #REQUIRED
    expires               CDATA         #REQUIRED
    integrityCheck        (true|false)  'false'
    baseURL               CDATA         #IMPLIED
    retransmitExpiration  CDATA         #IMPLIED>

<!ELEMENT component EMPTY>
<!ATTLIST component
    name          NMTOKEN     #IMPLIED
    src           CDATA       #REQUIRED
    location      CDATA       #IMPLIED
    expires       CDATA       #IMPLIED>

<!ELEMENT sync-data EMPTY>
<!ATTLIST sync-data
    startTime     NMTOKEN                    #REQUIRED
    field         (1|2)                      #IMPLIED
```



```
type          (C1|C2|C3|C4|T1|T2|T3|T4|XDS)     #REQUIRED
content       CDATA                             #REQUIRED>
```

Appendix B – Font Style

The font style is a space-delimited list of values (in any sequence) that are applied to the specific attribute for which the value is a valid type. In the case of the length typed attributes, font-size, and line-height, the first occurrence of a length will apply to the font-size and the second to the line-height. Font style is compatible with the font attribute specified by CSS2 [2].

A font has the following attributes: style, variant, weight, size, line-height, and family.

Font style indicates whether the element is rendered in a normal (Roman), italic, or oblique font style.

Font variant determines whether the element will be rendered in all uppercase letters with lowercase letters rendered as smaller uppercase letters.

Font weight sets the boldness of the element's font. The weight is represented as a numeric scale ranging from 100 to 900 in increments of 100. Normal weight is 400 and a standard bold font is 700. 900 is the darkest level, 100 is the lightest.

For use in ITVML, relative weights (lighter and bolder) are based on the normal font weight because elements are not contained by (children of) other elements.

Font size determines the height of the displayed text. Font size is represented by a list of constant strings ranging from xx-small to xx-large.

Relative sizes (larger and smaller) are based on the medium font size because elements are not contained by (children of) other elements.

Font size can be specified in points (pt) or ems (em). Ems is a multiplier applied to the medium font size. Font size can also be expressed as a percent of the medium font size. Table 8 indicates the size in points of the string constants.

Table 8 - Font Size of String Constraints

Constant	Point size	Ems
xx-small	14pt	.7em
x-small	16pt	.8em
small	18pt	.9em
medium	20pt	1em
large	24pt	1.2em
x-large	28pt	1.4em

xx-large	32pt	1.6em

Line height sets the spacing between lines of text. Normally, the line height matches the font size. A number value acts as a multiplier for the font-size of the current element. Line height may also be specified in points (pt), ems (em) or as a percentage of the font-size.

Font family is a prioritized list of font families to be used to render the text. One ore more font family names may be included in a comma-delimited list of attribute values. If a font family name consists of multiple words, the family name must be inside quotes.

Syntax Summary

font-style || *font-variant* || *font-weight* || *font-size* || *line-height* || *font-family*

font-style: normal | italic | oblique

font-variant: normal | small-caps

font-weight: bold | bolder | lighter | normal | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900

font-size: *absoluteSize* | *relativeSize* | *length* | *percentage*
- absoluteSize: xx-small | x-small | small | medium | large | x-large | xx-large
- relativeSize: larger | smaller
- length: described in points (pts) or ems (em)
- percentage: calculated based on the medium font-size

line-height: normal | *number* | *length* | *percentage*
- number: acts as a multiplier for the font-size
- length: described as points (pts) or ems (em)
- percentage: calculated based on size of element's font-size

font-family: *fontFamilyName* [, *fontFamilyName* [, ...]]
- fontFamilyName: *name-of-font-family* | *genericFamily*
- genericFamily: serif | sans-serif | cursive | fantasy | monospace

Appendix C – Regular Expressions

Regular expression notation is a pattern surrounded by forward slashes. The pattern consists of a search string and may contain a number of meta-characters. Meta-characters are used to represent ideas such as word boundaries and character types. Regular expression syntax used in ITVML adheres to *RegExp* syntax specified in ECMAScript [7].

Syntax Summary

Syntax summary: `/pattern/[i]`

Appending "i" to the regular expression indicates that case is ignored. For example /john/i matches "john", "John", and "JOHN".

Table 9shows meta-character notation supported in a pattern.

Table 9 - Meta-Character Notation for Regular Expressions

Character	Description	Example
\b	Word boundary	/\bto/ matches "together" /to\b/ matches "how-to" /\bto\b/ matches "to"
\B	Word non-boundary	/\Bto/ matches "stool" and "how-to" /to\B/ matches "stool" and "together" /\Bto\B/ matches "stool"
\d	Numeral 0 through 9	/\d\d/ matches "42"
\D	Non-numeral	/\D\D/ matches "to"
\s	Single white-space	/wolf\sman/ matches "wolf man"
\s	Single non-white-space	/wolf\Sman/ matches "wolf-man"
\w	Letter, numeral, or underscore	/1\w/ matches "1A"
\W	Not a letter, numeral, or underscore	/1\W/ matches "1%"
.	Any character except a new-line	/../ matches "C4"
[...]	Any character set in brackets	/J[aeiou]y/ matches "Jay" /[A-C]1/ matches "A1", "B1" and "C1"

[^...]	Negated character set	/J[^aeiu]y/ matches "Joy" /[^A-C]1/ matches "D1"
*	Zero or more times	/\d*/ matches "", "5", or "444"
?	Zero or one time	/\d?/ matches "" or "5"
+	One or more times	/\d+/ matches "5" or "444"
{n}	Exactly n times	/\d{2}/ matches "55"
{n,}	n or more times	/\d{2,}/ matches "555"
{n,m}	At least n, at most m times	/\d{2,4}/ matches "5555"
^	At beginning of a string or line	/^Simon/ matches "Simon says..."
$	At end of string or line	/Simon.$/ matches "Hi, Simon."
\	Protect a meta-character	/itvml\.org/ matches "itvml.org" but not "itvml-org"
|	Or operation	/red|green/ matches "red" and "green"
()	Group operation	/my (house|car)/ matches "my house" and "my car"

Examples:

`/^\d+$/` matches an integer ("2001").

`/^(+|-)?\d+$/` matches a signed integer ("-1", "1", or "+1").

`/^(((\d+(\.\d*)?)|((\d*\.)?\d+))$/` matches a floating-point number ("1.2", ".9", or "8").

`/^(((+|-)?\d+(\.\d*)?)|((\d*\.)?\d+))$/` matches a signed floating-point number ("-1.2", "1.2", ".9", or "8").

`/^[\(]?\d{3}[ -\.\)]? ?\d{3}[ -.]?\d{4}$/` matches a phone number with an area code making allowances for punctuation ("(303) 555-1234", "303-555-1234", or "3035551234").

`/^\d{5}(-\d{4})?$/` matches a zip code with an optional zip+4 ("80202" or "80202-2001").

`/^[\w-\.]+@[0-9a-z-\.]+\.([a-z]{2}|com|edu|gov|org)$/i` will match an email address with a valid top level domain ("itvml-editor@itvml.org").

References

[1] *Extensible Markup Language (XML)*, W3C Proposed Recommendation 10-February-1998, REC-xml-19980210, T. Bray, et al, February 10, 1998, http://www.w3.org/TR/REC-xml.

[2] B. Bos, H. W. Lie, C. Lilley, I. Jacobs, *Cascading Style Sheets, level 2, CSS2 Specification*. W3C Recommendation 12-May-1998, http://www.w3.org/TR/REC-CSS2/.

[3] J. Clark and S. Deach, eds, *Extensible Style Language (XSL)*, W3C Proposed Recommendation, 12 January 2000. http://www.w3.org/TR/xsl.

[4] J. Clark, *XSL Transformations (XSLT),* W3C Recommendation 16 November 1999, http://www.w3.org/TR/xslt.

[5] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels,* http://www.ietf.org/rfc/rfc2119.txt, March 1997.

[6] Advanced Television Enhancement Forum (ATVEF), *Enhanced Content Specification v1.1 r26*, http://www.atvef.com/library/spec1_1a.html, 1999.

[7] ECMA, *ECMA-262 ECMAScript Language Specification 3rd Edition,* http://www.ecma.ch/ecma1/stand/ecma-262.htm, 1999.

[8] N. Freed, Network Working Group, *Multipurpose Internet Mail Extensions(MIME): Format of Internet Message Bodies,* http://www.ietf.org/rfc/rfc2045.txt, 1996.